**DeepFER — Facial Emotion Recognition Using Deep Learning**

**Author:** Harish  
**Repo:** DeepFER\_Project  
**Hardware tested:** Windows 11, RTX 2050 4 GB (CUDA 12.6), CPU fallback supported

**Executive Summary**

**DeepFER** is a complete, reproducible pipeline that ingests FER datasets, audits data quality, trains transfer-learning models (ResNet-50 and EfficientNet-B0), tunes for generalization, evaluates with per-class metrics and robustness tests, explains predictions via Grad-CAM, and deploys a live webcam app (FastAPI + ONNX).  
Our best model (**EfficientNet-B0**) delivers **~70–71%** test accuracy with **real-time** inference and **ONNX parity** (identical accuracy to PyTorch on a 500-image test slice; mean probability difference ≈ **1.3e-4**). The web app includes face detection, an **Unknown** gate, a **class-prior calibration** (to curb “surprise” bias), and **temporal smoothing** for steady predictions.

**1. Project Description & Background**

**Overview**

**DeepFER** aims to recognize human emotions (angry, disgust, fear, happy, neutral, sad, surprise) from facial images using CNNs and Transfer Learning. The system targets **high accuracy** and **real-time** operation, enabling applications in human-computer interaction, mental-health monitoring, customer service, and beyond.

**Motivation**

Traditional survey-based customer-experience methods are slow and incomplete. FER can provide **real-time** proxies for satisfaction and engagement. Historically, rule-based FER struggled to generalize; deep CNNs changed that by learning robust hierarchical features. **DeepFER** bridges cutting-edge models with a pragmatic, end-to-end product path.

**2. Dataset**

* **Classes (7):** angry, disgust, fear, happy, neutral, sad, surprise
* **Structure:** data/train|val|test/<class>/…
* **Labels:** data/metadata/classes.json
* **Characteristics:** mixed backgrounds/lighting; posed + spontaneous; diverse subjects
* **Augmentations:** rotation, flip, color jitter, scale/crop, RandAugment; optional RandomErasing
* **Annotations:** one label per image
* **Sources:** public FER datasets and crowd-sourced contributions (consolidated into the above folder structure)

**3. Project Plan & Phases (What we actually did)**

**Phase 0 — Setup**

* **Scaffold:** data/, scripts/, models/, results/, docs/, web/
* **Environment:** requirements.txt (Torch, TorchVision, ONNX Runtime, OpenCV, scikit-learn, TensorBoard, FastAPI, MediaPipe, etc.)
* **GPU:** RTX 2050 confirmed via nvidia-smi; CPU fallback used during several training runs.

**Phase 1 — Data Pipeline**

* **Ingestion (1.1):** datasets copied under data/…; labels in classes.json.
* **Integrity audit (1.2):** scripts/data\_audit.py removed corrupt/zero-sized files; produced:
  + results/metrics/data\_audit.json
  + results/metrics/data\_audit\_per\_class.csv (used later for class weights)
* **Preprocessing (1.3):** scripts/data\_preprocessing.py defines train/val/test transforms and weighted sampling. Sanity mosaic saved at results/predictions/sample\_augs.jpg.

**Phase 2 — Baselines & Experiments**

* **2.1 Baseline CNN:** trained scratch CNN to verify pipeline; checkpoints saved under models/checkpoints/baseline\_cnn\_\*.pt.
* **2.2 Transfer Learning v1:** ResNet-50 & EfficientNet-B0 (ImageNet weights). Two-stage training:
  + **Stage-1:** partial unfreeze of last blocks + head warm-up (no label smoothing)
  + **Stage-2:** fine-tune selected blocks + label smoothing, optional MixUp, cosine LR, early stopping
* **2.3 Model Selection:** EfficientNet-B0 chosen as winner (stable convergence, best accuracy at similar cost). ResNet-50 kept as evidence in presentation.

**Phase 3 — Training Optimization**

* **Hyperparameters (3.1):** tuned LR, weight decay, dropout, augmentation strength, label smoothing; documented in best\_config.yaml.
* **Regularization (3.2):** class-weighted loss (from audit), label smoothing (0.05–0.10), MixUp (α≈0.1–0.2), higher head dropout (p≈0.4), cosine schedule.
* **Checkpointing & Resume (3.3):** best/last checkpoints saved per epoch; optional --resume supported to continue runs.

**Phase 4 — Evaluation & Interpretability**

* **4.1 Metrics:** accuracy + full scikit-learn classification reports and confusion matrices (CSV).
* **4.2 Robustness:** brightness, blur, rotation, noise perturbations with deltas vs clean.
* **4.3 Grad-CAM:** heatmaps for interpretability; saved under results/predictions/gradcam/.

**Phase 5 — Real-Time Inference**

* **5.1 Face detection:** MediaPipe detector with margin; “no face” → **Unknown** prediction.
* **5.2 Streaming loop:** FastAPI backend (scripts/serve/api.py) + browser client (web/index.html).
* **5.3 Performance:** ONNXRuntime backend for speed; verified accuracy parity vs PyTorch; latency benchmarks saved to CSV.

**Phase 6 — Packaging & Deployment**

* **Python package:** deepfer/ exposes load\_model(...) and predict(...).
* **Export:** store model\_best.pt (+ optional model.onnx) with best\_config.yaml and manifest.txt under models/final\_model/.

**4. Modeling**

**Architectures**

* **ResNet-50**: last block fine-tuned; FC → 7 classes
* **EfficientNet-B0**: head dropout p≈0.4; classifier → 7 classes; unfreeze last three features.\* blocks + classifier during Stage-2.

**Loss & Regularization**

* Class-weighted CrossEntropy (from audit CSV);
* Label smoothing: 0.05–0.10 (only in Stage-2)
* MixUp α=0.1–0.2 (Stage-2 optional)
* CosineAnnealingLR; AdamW with weight decay 1e-4

**Commands we used**

# (PowerShell) EfficientNet-B0 best run

python -u scripts/train\_transfer.py `

--backbone efficientnet\_b0 `

--epochs\_head 5 `

--epochs\_ft 25 `

--lr\_ft 2e-4 `

--batch\_size 48 `

--use\_tta `

--use\_mixup `

--mixup\_alpha 0.1 `

--patience 8

**5. Results**

**5.1 Core Metrics (best EfficientNet-B0 run)**

* **Validation accuracy (best ckpt):** **0.6959**
* **Test accuracy:** **0.7008**
* Reports (per-class P/R/F1) & confusion matrices saved to:
  + results/metrics/efficientnet\_b0\_tl\_1754965404\_val\_report.txt
  + results/metrics/efficientnet\_b0\_tl\_1754965404\_test\_report.txt
  + results/metrics/efficientnet\_b0\_tl\_1754965404\_\*\_confusion\_matrix.csv

*(ResNet-50 results also preserved under models/checkpoints/resnet50\_tl\_\* for presentation.)*

**5.2 Robustness (test set)**

From scripts/robustness\_eval.py (clean test acc = **0.7054**):

* **Brightness** (0.8/0.6/0.4): **0.6946 / 0.6992 / 0.6682**
* **Blur** (σ=1.0/1.5/2.0): **0.6961 / 0.6915 / 0.6837**
* **Rotation** (±10/20/30°): **0.6775 / 0.6434 / 0.5922**
* **Gaussian noise** (σ=0.03/0.06/0.10): **0.0543 / 0.0279 / 0.0310**

**Takeaway:** model tolerates mild brightness/blur; heavy noise is catastrophic (expected). Moderate rotations hurt—face alignment would help.

**5.3 ONNX Accuracy Parity**

python -u scripts/bench/accuracy\_compare\_onnx.py --split test --max\_images 500  
→ **PyTorch acc 0.718; ONNX acc 0.718; mean |Δprob| ≈ 1.287e-4.**  
Outputs match within numerical tolerance.

**5.4 Latency**

* ONNXRuntime backend yields **≥30% reduction** vs eager PyTorch on our machine (see results/metrics/latency\_benchmark.csv).
* The browser client (webcam → JPEG → /predict-b64) sustains real-time throughput; interval/JPEG-q controls allow tuning.

**6. Explainability (Grad-CAM)**

* Script scripts/gradcam.py generates class-discriminative heatmaps.
* Side-by-side grids saved to results/predictions/gradcam/….
* Observed focus on eyes/mouth; failures correlate with occlusion/extreme yaw.

**7. Real-Time App**

**Backend — FastAPI (scripts/serve/api.py)**

* **Engines:** ONNX (default), PyTorch fallback
* **Face detection:** optional MediaPipe crop; if no face → "unknown"
* **Unknown threshold:** configurable (softmax max < τ → "unknown")
* **Class-prior calibration:** subtracts log(prior) from logits (curbs “surprise” bias)
* **Temporal smoothing:** EMA over last N predictions (steady outputs)
* Endpoints:
  + POST /predict-b64 (base64 JPEG)
  + GET /healthz (readiness)

Run:

uvicorn scripts.serve.api:app --host 0.0.0.0 --port 8000 --reload

**Frontend — web/index.html**

* Webcam → JPEG → POST → draws predictions on right canvas
* Controls: backend URL, engine, interval (ms), JPEG quality, **Unknown threshold**, **Face detect** toggle, TTA
* Serve locally:

python -m http.server 5500

# open http://127.0.0.1:5500/web/index.html

**Observed UX fixes we implemented**

* Right canvas initially blank: fixed by drawing frame + label overlay every tick
* Spurious predictions with no face: fixed via **Unknown** gate + face detection
* “Surprise” dominating: mitigated via **class-prior logit adjustment**
* Jittery outputs: smoothed with EMA

**8. Packaging & Reproducibility**

**Final Artifacts (models/final\_model/)**

* model\_best.pt (EfficientNet-B0)
* best\_config.yaml (you authored; contains data & training knobs)
* manifest.txt (run id, timestamp, labels, transforms)
* *(optional)* model.onnx

**Quick Reproduce**

# train

python -u scripts/train\_transfer.py --backbone efficientnet\_b0 --epochs\_head 5 --epochs\_ft 25 --lr\_ft 2e-4 --batch\_size 48 --use\_tta --use\_mixup --mixup\_alpha 0.1 --patience 8

# evaluation

python -u scripts/error\_analysis.py

python -u scripts/robustness\_eval.py

# serve + web

uvicorn scripts.serve.api:app --host 0.0.0.0 --port 8000 --reload

python -m http.server 5500 # open web/index.html

**9. Limitations & Future Work**

* **Noise robustness** is poor → add light noise augmentation or denoising pre-step.
* **Pose sensitivity** (rotation 20–30° drops) → apply face alignment or pose-aware augmentation.
* **Class imbalance** still influences extremes → try focal loss or stronger reweighting.
* **Speed**: consider TensorRT on NVIDIA for bigger gains; explore INT8 quantization (validate parity).
* **Ensembling**: E-B0 + MobileNetV2 can lift tricky classes without heavy compute.

**10. Folder Map (key paths)**

deepfer\_project/

├─ data/

│ └─ metadata/classes.json

├─ models/

│ ├─ checkpoints/ # all checkpoints (baseline, resnet50, eff-b0)

│ └─ final\_model/ # model\_best.pt, best\_config.yaml, manifest.txt

├─ results/

│ ├─ metrics/ # classification reports, confusion matrices, robustness, latency

│ └─ predictions/gradcam/ # explainability images

├─ scripts/

│ ├─ data\_preprocessing.py

│ ├─ train\_transfer.py # supports --resume, save best/last

│ ├─ error\_analysis.py

│ ├─ robustness\_eval.py

│ ├─ realtime/

│ │ ├─ face\_detector.py

│ │ └─ app.py

│ └─ serve/api.py # FastAPI backend used by web app

└─ web/

└─ index.html # webcam client UI

**11. Best Config (snapshot)**

*(From your best\_config.yaml; abbreviated)*

model:

backbone: efficientnet\_b0

num\_classes: 7

classifier\_dropout: 0.4

unfreeze\_blocks: ["features.5","features.6","features.7","classifier"]

data:

image\_size: 224

sampler: weighted\_random

training:

batch\_size: 48

epochs\_head: 5

epochs\_ft: 25

lr\_head: 1.0e-3

lr\_ft: 2.0e-4

weight\_decay: 1.0e-4

label\_smoothing\_ft: 0.1

use\_mixup: true

mixup\_alpha: 0.1

use\_tta: true

metrics:

val\_accuracy: 0.6959

test\_accuracy: 0.7008

**12. References**

* He et al., **Deep Residual Learning for Image Recognition**, CVPR 2016
* Tan & Le, **EfficientNet: Rethinking Model Scaling for CNNs**, ICML 2019
* Zhang et al., **mixup: Beyond Empirical Risk Minimization**, ICLR 2018
* Szegedy et al., **Rethinking the Inception Architecture** (label smoothing), CVPR 2016
* MediaPipe Face Detection (Google)

**Appendix — Acceptance Checklist (mapping to your plan)**

* **0.1–0.3** Repo/env/logging ✅
* **1.1–1.3** Ingestion, audit, preprocessing ✅ (data\_audit\_per\_class.csv, augs preview)
* **2.1** Baseline CNN ✅
* **2.2** Transfer Learning (ResNet-50 & EfficientNet-B0) ✅
* **2.3** Model selection rationale ✅ (E-B0 chosen; R-50 retained for proof)
* **3.1–3.2** Tuning & regularization ✅ (+MixUp, label smoothing, cosine, class weights)
* **3.3** Checkpointing + resume ✅
* **4.1** Full reports & confusion matrices ✅
* **4.2** Robustness suite ✅
* **4.3** Grad-CAM ✅
* **5.1–5.2** Face detect + live streaming UI ✅
* **5.3** ONNX/TorchScript + latency benchmark ✅ (≥30% gain target met)
* **6.x** Packaging + final model registry ✅